LAUTERBACH
DEVELOPMENT TOOLS

78K0R/RL78 Debugger

MANUAL

78K0R/RL78 Debugger

78K0R/RL78 Debugger

Version 06-Jun-2024

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Training Basic Debugging”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **“OS Awareness Manuals”** (rtos_<*os*>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Demo and Start-up Scripts

To search for PRACTICE scripts, do one of the following in TRACE32 PowerView:

- Type at the command line: **WELCOME.SCRIPTS**
- or choose **File** menu > **Search for Script**.

  You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.

You can also manually navigate in the `~~/demo/78k0r/` and `~~/demo/rl78/`subfolders of the system directory of TRACE32.

Warning

WARNING:

To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF.

Recommendation for the software start:

1. Disconnect the Debug Cable from the target while the target power is off.
2. Connect the host system, the TRACE32 hardware and the Debug Cable.
3. Power ON the TRACE32 hardware.
4. Start the TRACE32 software to load the debugger firmware.
5. Connect the Debug Cable to the target.
6. Switch the target power ON.
7. Configure your debugger e.g. via a start-up script.

Power down:

1. Switch off the target power.
2. Disconnect the Debug Cable from the target.
3. Close the TRACE32 software.
4. Power OFF the TRACE32 hardware.

General Notes/Target Design Requirements/Recommendations

General

- The Lauterbach TRACE32 debugger for 78K0R/RL78 is an on-chip debugging tool (OCD). It uses the debug function implemented in the target CPU via a serial communication link (no JTAG).
- Debug functionality uses a monitor program in flash memory. Due to that some resources are used by the debugger exclusively (see limitations). Depending on the compiler besides the activation of debug option some setting have to be done (refer the compiler manual).

Target Design Requirements

- Locate the debug connector as close as possible to the processor to minimize the capacitive influence and cross coupling of noise onto the signals. Do not put any capacitors (or RC combinations) on the TOOL0/TOOL1 lines.
- Reduce the cable length between CPU and Lauterbach connector to a minimum. Best results will be provided, if a adequate connector will be foreseen directly on the target board.
- A Pull-up resistor of about 1,5 kOhm has to adapted between TOOL0 and VDD. This could be done on the target board or via an adaptation in the debug cable.

Limitations

- The last block of internal flash memory is blocked for writing. For a block size of 1 kB (Fx3) and 256 kB of total code flash memory (78F1845) the address range 0x3FC00--0x3FFFF has to be declared as “NOP”.
- Debug monitor consumes 6 bytes of stack area, right before stack pointer SP. For example for a SP value of 0xFE00, the memory range 0xFFDFA--0xFFDFF will be written by the debug monitor before the transition RUN to BREAK.

  For accurate debugging **SP must always be 6 byte higher than the start address of RAM area**. To avoid impacts from debugger to user data, designated stack area should be least 6 bytes bigger than nominally necessary.
- The first **4 bytes (0x00000--0x00003)** of program memory are reserved for debugging issues and must not be changed.
- **21 bytes of program memory space (0x000C3--0x000D7) is reserved for debug purposes** (refer the target CPU manual for any details).

  At 0x000CE the user reset vector is placed. User program will start from 0x000D8 or at higher address.
- The Pins of **TOOL0 and TOOL1 (Port 4)** are not usable during debugging.

Quick Start

Starting up the debugger is done by the following steps:

1. Select the device prompt B: for the TRACE32 ICD-Debugger, if the device prompt is not active after starting the TRACE32 software.,

```
b::
```

The device prompt `B::` is normally already selected in the TRACE32 command line. If this is not the case, enter `B:` to set the correct device prompt. A **RESet** command is useful if you do not start directly after booting the TRACE32 development tool.

2. Select the CPU derivate to load the specific settings.

```
SYStem.CPU 78F1845
```

The default value for SYStem.CPU is “78K0R”, which is not a real existing derivate. This means that the debugger tries an automatic detection of the connected derivate.

The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Enter debug mode.

```
SYStem.Up
```

4. Declare size and type of **FLASH** memory is recommended doing via script.

```
DO ~~/demo/78k0r/flash/78k0r_*.cmm
```

Select the adequate PRACTICE script for the connected target. It will set up the flash memory to allow writing and setting of software-breakpoints.

If not loaded a program during the execution of the flash script, this can be done as follows:

5. Load the program.

```
Data.LOAD sieve.dbg          ; sieve.dbg is the file name
```

The format of the **Data.LOAD** command depends on the file format generated by the compiler. Without any specific format (like in this example), TRACE32 tries to detected the correct format available for 78K0R-debugger depending on the specified file.

A detailed description of the **Data.LOAD** command and all available options is given in the**”General Reference Guide”**.

A typical start sequence is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

```
B::                               ; Select the ICD-Debugger device prompt

WinCLEAR                          ; Clear all windows

SYStem.CPU 78F1845                ; Select the CPU derivate type

SYStem.Up                         ; Reset the target and enter debug mode

Data.LOAD.UBROF sieve.d26         ; Load the application (here IAR-
                                  ; Compiler)

Register.Set PC main              ; Set the PC to function main

Register.Set SP 0xfe20            ; Set the stack pointer to address
                                  ; 0xfe20

PER.view                          ; Open a window for the special
                                  ; function registers and peripherals *)

List.Mix                          ; Open source code window            *)

Register.view /SpotLight          ; Open register window               *)

Frame.view /Locals /Caller        ; Open the stack frame with
                                  ; local variables                    *)

Var.Watch var1 var2               ; Open watch window for variables    *)

Var.Local                         ; Open window with local variables   *)

Break.Set 0xffc00 /Program        ; Set software breakpoint to address
                                  ; 0xffc00 (address in RAM)

Break.Set 0x100 /Onchip           ; Set on-chip BP to address 0x100
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

Troubleshooting

Communication between Debugger and Processor can not be established

Typically the **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages like “debug port fail” or “debug port time out” while executing this command, this may have the reasons below. “target processor in reset” is just a follow-up error message. Open the **AREA.view** window to see all error messages.

- The target has no power or the debug cable is not connected to the target. This results in the error message “target power fail”.
- You did not select the correct core type **SYStem.CPU** *<type>*.
- The target is in an unrecoverable state. Re-power your target and try again.
- The default debug clock speed is too fast, especially if the target is connect to debugger by a long cable. Reduce the communication speed with **SYStem.DebugClock** command and optimize the speed when you got it working.
- The CPU has no clock.
- The CPU is kept in reset.
- There is a watchdog which needs to be deactivated.
- Your target needs special debugger settings. Check the directory \demo\78k0r if there is an suitable script file *.cmm for your target.

FAQ

Please refer to https://support.lauterbach.com/kb.

78K0R/RL78 Specific Implementations

Breakpoints

Two types of breakpoints are available for 78K0R/RL78 architecture: Software breakpoints and one on-chip breakpoint.

Software Breakpoints

To set a software breakpoint, before resuming the CPU, the debugger replaces the instruction at the breakpoint address with a breakpoint code instruction.

There is no restriction in the number of software breakpoints. But it must be considered that by the usage of software breakpoint flash memory will be change, if program is run in flash memory.

On-chip Breakpoints

If on-chip breakpoints are used, the resources to set the breakpoints are provided by the CPU. To set breakpoints on code in read-only memory, only the on-chip instruction address breakpoints are available. With the command **MAP.BOnchip** *<range>* it is possible to declare memory address ranges for use with on-chip breakpoints to the debugger.

Besides the restricted number of on-chip breakpoint, the biggest disadvantage of 78K0R/RL78 architecture is that the breakpoints is effected **after execution**. It means that not only the corresponding opcode will be executed, instead the whole instruction pipeline will be executed eventually.

- **On-chip breakpoints:** Total amount of available on-chip breakpoints.
- **Instruction breakpoints:** Number of on-chip breakpoints that can be used to set program breakpoints into ROM/FLASH/EPROM.
- **Read/Write breakpoints:** Number of on-chip breakpoints that can be used as Read or Write breakpoints.
- **Data breakpoint:** Number of on-chip data breakpoints that can be used to stop the program when a specific data value is written to an address or when a specific data value is read from an address.

Core type:	On-chip Breakpoints	Instruction Address Breakpoints	Data Address Breakpoints	Data Value Breakpoints
78K0R/RL78	1 Instruction or 1 Read/Write	1 breakpoint	1 single address above 0xF0000	1 single value -- or -- 1 value mask

You can check your currently set breakpoints with the command **Break.List.**

If no more on-chip breakpoints are available you will get an error message on trying to set a new on-chip breakpoint.

Breakpoints on Data Addresses and Data Values

Breakpoints on data addresses are bound to several conditions:

1. The source of the data access (read and/or write) must be the CPU, as the data address breakpoints are part of the CPU. Any other accesses from on-chip or off-chip peripherals (DMA etc.) will not be recognized by the data address breakpoints.

2. The data being targeted must be qualified by an address in memory in the range 0xF000--0xFFFFF. It is not possible to set a data address breakpoint to GPR, SPR etc.

3. The CPU stops the application execution only if the address and the **access width** in the field DATA of the **Break.Set** window match. An empty access width is equal to standard width, which is Word.

4. Per default the break will be done independently of the value (empty DATA field of **Break.Set** window).

Example for Standard Breakpoints

Assume you have a target (78F1845) with:

- Code flash memory from `0x00000--0x3ffff`
- RAM from `0xfbf00--0xffedf`

The following standard breakpoint combinations are possible without activated auto flash mode:

1. Unlimited breakpoints in RAM and one breakpoint in ROM/FLASH

```
Break.Set 0xfcf00 /Program            ; Software breakpoint 1

Break.Set 0xfd000 /Program            ; Software breakpoint 2

Break.Set addr /Program               ; Software breakpoint 3

Break.Set 0x100 /Program              ; On-chip breakpoint
```

2. Unlimited breakpoints in RAM and one breakpoint on a read or write access

```
Break.Set 0xfcf00 /Program            ; Software breakpoint 1

Break.Set 0xfd000 /Program            ; Software breakpoint 2

Break.Set addr /Program               ; Software breakpoint 3

Break.Set 0xfef47 /Write              ; On-chip breakpoint
```

With activated auto flash mode even in code flash memory unlimited breakpoints are allowed:

1. Unlimited breakpoints in ROM/FLASH

```
Break.Set 0x00200 /Program              ; Software breakpoint 1

Break.Set 0x01000 /Program              ; Software breakpoint 2

Break.Set addr /Program                 ; Software breakpoint 3

Break.Set 0x00100 /Program              ; On-chip breakpoint
```

Runtime Measurement

The command **RunTime** allows run time measurement based on polling the CPU run status by software. Therefore the result will be about few milliseconds higher than the real value.

If the signal DBGACK on the JTAG connector is available, the measurement will automatically be based on this hardware signal which delivers very exact results.

Memory Classes

Even if the 78K0R/RL78 as Von Neumann architecture has a linear memory space. The following 78K0R/RL78 specific memory classes are available:

Memory Class	Description
P	Program Memory
D	Data Memory
VM	Virtual Memory (memory on the debug system)
E	Run-time memory access (see **SYStem.CpuAccess** and **SYStem.MemAccess**)

SYStem.CONFIG.state

Display target configuration

Format:	**SYStem.CONFIG.state** [**/**<*tab*>]
<*tab*>:	**DebugPort**

Opens the **SYStem.CONFIG.state** window, where you can view and modify most of the target configuration settings. The configuration settings tell the debugger how to communicate with the chip on the target board and how to access the on-chip debug and trace facilities in order to accomplish the debugger's operations.

Alternatively, you can modify the target configuration settings via the TRACE32 command line with the **SYStem.CONFIG** commands. Note that the command line provides *additional* **SYStem.CONFIG** commands for settings that are *not* included in the **SYStem.CONFIG.state** window.

DebugPort	Informs the debugger about the debug connector type and the communication protocol it shall use.

SYStem.CONFIG Configure debugger according to target topology

Format:	**SYStem.CONFIG** *<parameter>*
<parameter>:	**DEBUGPORT** [**DebugCable0** | …] **DEBUGPORTTYPE** [**TOOL0** | **TOOL0+1**] **TriState** [**ON** | **OFF**]

DEBUGPORT Specifies which probe cable shall be used.

DEBUGPORTTYPE Specifies the used debug port type.

- **TOOL0**: 1-wire mode, mandatory for RL78
- **TOOL0+1**: 2-wire mode, recommended for 78K0R

TriState TriState does not hold the target in reset state during **SYStem.Down** state. This allows to execute the flash program of the target without any debugger control and without the necessity to disconnect the target from the debugger electrically.

On a **SYStem.Up** the target will be reset.

Default: OFF.

SYStem.CPU Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**AUTO** | **78K0R** | **RL78** | **78F***<xxxx>* | … | **R5F1***<xxxx>*

Selects the processor type. If your chip is not listed, try the default value or contact technical support.

AUTO (default) Automatic detection of the CPU (78K0R and RL78)

78K0R Automatic detection of a 78K0R-CPU

RL78 Automatic detection of a RL78-CPU

78F*<xxxx>*	Sets the parameters (RAM size, Flash size, PER file, ...) defined by the selected CPU (78K0R derivates)
R5F1*<xxxx>*	Sets the parameters (RAM size, Flash size, PER file, ...) defined by the selected CPU (RL78 derivates)

SYStem.DebugClock — Set debug clock frequency

Format:	**SYStem.DebugClock** <*rate*>
<*rate*>:	**1/4** … **1/256**

Selects the frequency for the debug interface. The value represents the quotient of the division: Debug communication frequency divided by the frequency of TOOL1 (half of CPU frequency). As long as no problems occur it is recommend not change the default value.

In the case of unstable debug connection between debugger and target, lower values (towards 1/256) could help. Higher values (towards 1/4) effect in a faster debug communication, but the higher frequencies on the TOOL0 cable could lead to unstable debug connection.

Default value: 1/16

Example: Target frequency is 4 MHz, DebugClock is 1/16, transmission rate (nominal) is:
0.5 * 4 MHz / 16 = 125 kHz = 125 kbit/s

SYStem.LOCK — Lock and tristate the debug port

Format:	**SYStem.LOCK** [**ON** | **OFF**]

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the **SYStem.LOCK** command is to give debug access to another tool.

Format:	**SYStem.MemAccess** <*mode*> **SYStem.ACCESS** (deprecated)
<*mode*>:	**Enable** **StopAndGo** **Denied**

Default: Denied.

If **SYStem.MemAccess** is not **Denied**, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program.

Enable CPU (deprecated)	A run-time memory access is made without CPU intervention while the program is running. This is only possible on the instruction set simulator.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.
Denied	No memory access is possible while the CPU is executing the program.

If specific windows that display memory or variables should be updated while the program is running, select the memory class E: or the format option %E.

```
Data.dump E:0x100

Var.View %E first
```

SYStem.Mode Establish the communication with the target

Format:	**SYStem.Mode** <*mode*> **SYStem.Down** **(alias for SYStem.Mode Down)** **SYStem.Up** **(alias for SYStem.Mode Up)**
<*mode*>:	**Down** | **Attach** | **NoDebug** | **Go** | **Up**

Select target reset mode.

Down Disables the debugger. The state of the CPU remains unchanged.

NoDebug Resets the target with debug mode disabled. In this mode no debugging is possible. The CPU state keeps in the state of NoDebug.

Go Resets the target with debug mode enabled and prepares the CPU for debug mode entry. After this command the CPU is in the **SYStem.Up** mode and running. Now, the processor can be stopped with the break command or any break condition.

Up Resets the target and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging. All register are set to the default value.

NOTE: The system modes Attach and StandBy are not available for this architecture.

SYStem.state Display SYStem.state window

Format:	**SYStem.state**

Displays the **SYStem.state** window of the 78K0R/RL78 debugger.

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

Format: **SYStem.Option.IMASKHLL** [**ON** | **OFF**]

Default: OFF.

If enabled, the interrupt mask bits of the cpu will be set during HLL single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

SYStem.Option.KEYCODE Define 10 byte on-chip security ID

Format: **SYStem.Option.KEYCODE** <*sec_id*>

The default value for a plain CPU is set during **SYStem.Up**. If flash contains already a program with activated security ID the correct ID has to be insert before the **SYStem.Up** command, otherwise it is impossible to establish a debug connection.

The key code will automatically updated if a program will be loaded with activated security ID option.

<*sec_id*>	Any ID code of 10 bytes (20 hexadecimals).

SYStem.Option.ResetMASK Disable internal reset

Format: **SYStem.Option.RESETMASK** [**ON** | **OFF**]

Default: OFF.

If enabled, all internal resets are masked out and do not take any effects. This is for example useful when internal watch dog is activated.

SYStem.Option.SerialFreeze Stops serial transmissions during break

Format: **SYStem.Option.SerialFreeze** [**ON** | **OFF**]

Default: OFF.

If activated serial transmission will be stopped if the target is stopped.

SYStem.Option.TimerFreeze Stops all internal timers during break

Format: **SYStem.Option.TimerFreeze** [**ON** | **OFF**]

Default: OFF.

If activated the counting operations all internal timers will be stopped if the target is stopped.

CPU specific TrOnchip Commands

The **TrOnchip** command group is not available for the 78K0R/RL78 debugger.

Debug Connection

Pinout of the 16-pin Debug Cable:

Signal	Pin	Pin	Signal
VSS	1	2	RESET-
TOOL0	3	4	VDD
N/C	5	6	N/C
N/C	7	8	N/C
N/C	9	10	N/C
N/C	11	12	N/C
N/C	13	14	FLMD0
TRESET-	15	16	TOOL1

For details on logical functionality, physical connector, alternative connectors, electrical characteristics, timing behavior and printing circuit design hints refer to the application note **“Arm Debug and Trace Interface Specification”** (app_arm_target_interface.pdf).